DTIC FILE COPY

LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-428

AD-A222 823

THE NEED FOR HEADERS: AN IMPOSSIBILITY RESULT FOR COMMUNICATION OVER UNRELIABLE CHANNELS

Alan Fekete
Nancy A. Lynch

DTIC
ELECTE
JUN 19 1990
S D
E

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

March 1990

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

90 06 18 210

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MIT/LCS/TM - 428	5. MONITORING ORGANIZATION REPORT NUMBER(S) N00014-83-K-0125 / N00014-89-J-1988
6a. NAME OF PERFORMING ORGANIZATION MIT Lab for Computer Science 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Office of Naval Research/Dept. of Navy
6c. ADDRESS (City, State, and ZIP Code) 545 Technology Square Cambridge, MA 02139	7b. ADDRESS (City, State, and ZIP Code) Information Systems Program Arlington, VA 22217
8a. NAME OF FUNDING/SPONSORING ORGANIZATION DARPA/DOD 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code) 1400 Wilson Blvd. Arlington, VA 22217	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)

The Need for Headers: An Impossibility Result for Communication Over Unreliable Channels.

12. PERSONAL AUTHOR(S)
Alan Fekete, Nancy A. Lynch

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Technical	FROM _____ TO _____	March 1990	24

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	data link protocol, communication protocol, headers, impossibility proof

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Abstract

It is proved that any protocol that constructs a reliable data link service using a physical channel service necessarily includes in the packets some header information that enables the protocol to treat different packets differently. The physical channel considered is permitted to lose, but not reorder or duplicate packets. The formal framework used for the proof is the ~~I/O~~ input/output automaton model.

Keywords: concurrent programming

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	Unclassified
22a. NAME OF RESPONSIBLE INDIVIDUAL Judy Little	22b. TELEPHONE (Include Area Code) (617) 253-5894
	22c. OFFICE SYMBOL

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted. All other editions are obsolete.

GPO U.S. Government Printing Office: 1985-607-047

The Need for Headers: An Impossibility Result for Communication over Unreliable Channels

Alan Fekete
Department of Computer Science
University of Sydney
NSW 2006
AUSTRALIA

Nancy Lynch
MIT Lab for Computer Science
545 Technology Square
Cambridge, MA 02139

May 15, 1990

Abstract

It is proved that any protocol that constructs a reliable data link service using a physical channel service necessarily includes in the packets some header information that enables the protocol to treat different packets differently. The physical channel considered is permitted to lose, but not reorder or duplicate packets. The formal framework used for the proof is the I/O automaton model.

Keywords: data link protocol, communication protocol, headers, impossibility proof

1 Introduction

Formal models of concurrent programming have been advanced as a suitable foundation for providing rigorous verification of protocols against specifications. A very different use is to give proofs *of impossibility results:* showing that no protocol can possibly solve a particular problem. The features of a formal model that are necessary to support impossibility proofs are not necessarily the same as those that make verification easy. A discussion of these features can be found in [11]. In this paper, we use the I/O automaton formal model (see [10] for a general exposition of the model) to provide a proof that any protocol that provides a data link service by using a physical channel service, necessarily includes in the packets some header information that enables the protocol to treat different packets differently. The interest of this work, we believe, is not so much in the result (no one ever suggested using a protocol without header information) but rather in the way this formal model (which has proved successful in verifying quite complicated protocols, as in [9, 15, 3, 4, 6, 14]) can be used to show the nonexistence of protocols with certain properties.

The authors were supported in part by the National Science Foundation under grant CCR-86-11442, by the Office of Naval Research under contract N00014-85-K-0168 and by the Defense Advanced Research Projects Agency under contracts N00014-83-K-0125 and N00014-89-J-1988.

There has recently been a lot of research in the distributed computing theory research community into the possibility of constructing a reliable message transmission service using an underlying unreliable packet transmission service (see [8, 1, 16, 12], for example). Most of this work has addressed the case where the physical channel is especially unreliable, in that it can lose packets and also deliver packets out of order. In these cases the natural protocol, due to Stenning ([13]) places each message into a packet with a sequence number as header, and repeatedly sends the packet until its receipt has been acknowledged. The difficulty with this protocol is that the sequence numbers increase without bound, and the papers mentioned above explore the possibility of using a fixed size header. By contrast, in this paper we consider using a FIFO (but possibly lossy) physical channel. There are many protocols known for this situation, most being variants on the Alternating Bit protocol [2], in which packets and acknowledgments contain a single bit header. We show that this header is needed, in that there is no protocol that solves the same problem without using some header to distinguish between packets. A key modeling issue is how to measure the existence of a header in an *arbitrary* protocol, without assuming a particular structure (such as [sequence-number,message]) for the packets. The definitions we use are adapted (and simplified) from those in [8, 5].

In Sections 2-4, we show how we model the different service specifications and the construction of an arbitrary protocol to provide a data link service using two physical channels. In Section 5 we define the specific physical channels we will use in the main result. In Section 6 we prove a result that includes much of what we want, while avoiding the more subtle modeling issues: we show the impossibility of implementing a data link service using identical packets in each direction. In Section 7 we discuss how to define the headers used by an arbitrary protocol, and finally we present the impossibility result. A summary of the I/O Automaton model is given in the Appendices.

2 The Physical Layer

The physical layer is the lowest layer in the OSI Reference Model hierarchy, and is implemented directly in terms of the physical transmission media. A standard interface to the physical layer permits implementation of the higher layers independently of the transmission media.

In a typical setting, a physical layer interacts with higher layers at two endpoints, a "transmitting station" and a "receiving station". The physical layer receives messages called "packets" from the higher layer at the transmitting station, and delivers some of the packets to the higher layer at the receiving station. The physical layer can lose packets. While it is also possible for packets to be corrupted by the transmission medium, we assume that the physical layer masks such corrupted packets using error-detecting codes. Thus, the only faulty behavior we consider is loss of packets.

In this section, we give a specification for physical layer behavior; in particular, we specify a channel that ensures FIFO delivery of packets. It is convenient to parameterize the specification by an ordered pair (t, r) of names for the transmitting and receiving stations, and by an alphabet P of legal *packets*. The specification will be given by a schedule module, denoted by $PL - FIFO^{t,r,P}$.

$PL - FIFO^{t,r,P}$ has the action signature illustrated in Figure 1 and given formally as follows.

Input actions:

$send\_pkt^{t,r}(p)$, $p \in P$

Output actions:

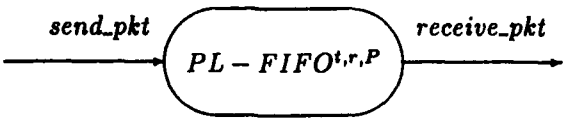

Figure 1: The Physical Layer

$receive\_pkt^{t,r}(p), p \in P$

Accession For
NTIS GRA&I
DTIC TAB
Unannounced
Justification
By
Distribution/
Availability Codes
Dist | Avail and/or Special
A-1
QUALITY INSPECTED 1

There are no internal actions. The $send\_pkt^{t,r}(p)$ action represents the sending of packet p on the physical channel by the transmitting station, and the $receive\_pkt^{t,r}(p)$ represents the receipt of packet p by the receiving station. We will refer to the actions in $acts(PL - FIFO^{t,r,P})$ as *physical layer actions* (for (t,r) and P).

In order to define the sets of schedules for the schedule module $scheds(PL - FIFO^{t,r,P})$, we define first a collection of properties, reflecting the operation of a "good" physical channel. The properties are defined with respect to $\beta = \pi_1\pi_2\ldots$ a (finite or infinite) sequence of physical layer actions, and a *correspondence relation*, a binary relation between the $send\_pkt^{t,r}$ events and the $receive\_pkt^{t,r}$ events in β. The correspondence relation is intended to model the association that can be set up between the event modeling the sending of a packet, and the event modeling the receipt of the same packet. Complications are caused by the fact that the same data might be sent repeatedly, and so the sending of two such identical packets is modeled by two occurrences of the same action $send\_pkt^{t,r}(p)$.

(PL1)
1. If an event $\pi_i = receive\_pkt^{t,r}(p)$ corresponds to an event $\pi_j = send\_pkt^{t,r}(q)$, then $p = q$, and also $j < i$, that is, the event π_j precedes π_i in β.
2. Each $receive\_pkt^{t,r}(p)$ event corresponds to exactly one $send\_pkt^{t,r}(p)$ event.
3. Each $send\_pkt^{t,r}(p)$ event corresponds to *at most* one $receive\_pkt^{t,r}(p)$ event.

Thus, when (PL1) is satisfied, any $receive\_pkt^{t,r}(p)$ in β will have a corresponding $send\_pkt^{t,r}$ event.

The next property we define is the FIFO property. It says that those packets that are delivered have their *receive_pkt* events occurring in the same order as their *send_pkt* events. Note that (PL2) may be true even if a packet is delivered and some packet sent earlier is not delivered; there can be gaps in the sequence of delivered packets representing lost packets.

(PL2) (FIFO) Suppose that the event $\pi_i = send\_pkt^{t,r}(p)$ in β corresponds to the event $\pi_j = receive\_pkt^{t,r}(p)$, and $\pi_k = send\_pkt^{t,r}(p')$ corresponds to $\pi_l = receive\_pkt^{t,r}(p')$. Then $i < k$ if and only if $j < l$.

So far, the properties listed have been safety properties, that is, when they hold for a sequence they also hold for any prefix of that sequence. The final property is a liveness property. It says that if repeated send events occur, then eventually some packet is delivered.

(PL3) If infinitely many $send\_pkt^{t,r}$ actions occur in β, then infinitely many $receive\_pkt^{t,r}$ actions occur in β.

Now we define the schedule module $PL\text{-}FIFO^{t,r,P}$. We have already defined $sig(PL\text{-}FIFO^{t,r,P})$. Let $scheds(PL\text{-}FIFO^{t,r,P})$ be the set of sequences β of physical layer actions for which there exists a correspondence such that (PL1), the FIFO condition (PL2), and (PL3), are all satisfied for β and that correspondence. A *FIFO physical channel* from t to r is any I/O automaton that solves $PL\text{-}FIFO^{t,r,P}$.

In a "real-world" implementation of a physical channel using a physical transmission medium, (PL3) would not be guaranteed with absolutely certainty, but rather with extremely high probability. In practice, this probability is usually sufficiently high to justify our decision to ignore in the formal model the small likelihood that no packets ever get delivered on an active channel, just as we have neglected the small probability of "real-world" channels corrupting a packet undetectably.

3 The Data Link Layer

The data link layer is the second lowest layer in the hierarchy, and is implemented using the services of the physical layer. Generally, it is implemented in terms of two physical channels, one in each direction. It provides a reliable one-hop message delivery service, which can in turn be used by the next higher layer.

We again assume that there are two endpoints, a "transmitting station" and a "receiving station". The data link layer receives messages from the higher layer at the transmitting station, and delivers them at the receiving station. The data link layer guarantees that every message that is sent is eventually received. Furthermore, the order of the messages is preserved.

In this section, we give a specification for data link layer behavior, as a parameterized schedule module $DL^{t,r,M}$, where M is an alphabet of legal *messages*. The development is very similar to that for the physical layer; in fact, the only significant difference is between the liveness conditions (DL3) and (PL3). The action signature $sig(DL^{t,r,M})$ is illustrated in Figure 2, and is given formally as follows.

Input actions:
$send\_msg^{t,r}(m)$, $m \in M$
Output actions:
$receive\_msg^{t,r}(m)$, $m \in M$

There are no internal actions. The $send\_msg^{t,r}(m)$ action represents the sending of message m on the data link by the transmitting station, and the $receive\_msg^{t,r}(m)$ represents the receipt of message m by the receiving station. We will refer to the actions in $acts(DL^{t,r,M})$ as *data link layer actions*.

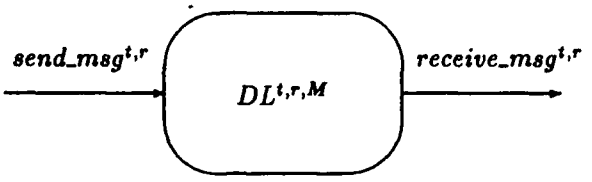

Figure 2: The Data Link Layer

In order to define the set $scheds(DL^{t,r,M})$, we again define a collection of auxiliary properties. They are defined for a sequence $\beta = \pi_1\pi_2\ldots$ of data link layer actions and a correspondence relationship, a binary relation between the $send\_msg^{t,r}$ events and the $receive\_msg^{t,r}$ events in β. The first property is analogous to (PL1) and gives elementary requirements on the correspondence.

(DL1)
1. If an event $\pi_i = receive\_msg^{t,r}(m)$ corresponds to an event $\pi_j = send\_msg^{t,r}(n)$, then $m = n$, and also $j < i$, that is, the event π_j precedes π_i in β.
2. Each $receive\_msg^{t,r}(m)$ event corresponds to exactly one $send\_msg^{t,r}(m)$ event.
3. Each $send\_msg^{t,r}(m)$ event corresponds to *at most* one $receive\_msg^{t,r}(m)$ event.

The next property is the FIFO property; it guarantees that the messages sent are received in the same order.

(DL2) (FIFO) Suppose that the event $\pi_i = send\_msg^{t,r}(m)$ in β corresponds to $\pi_j = receive\_msg^{t,r}(m)$, and $\pi_k = send\_msg^{t,r}(m')$ corresponds to $\pi_l = receive\_msg^{t,r}(m')$. Then $i < k$ if and only if $j < l$.

Finally, we have the data link layer liveness property. It says that all messages that are sent are eventually delivered. This property expresses the reliability of the message delivery guaranteed by the data link layer.

(DL3) If π is a $send\_msg^{t,r}(m)$ event occurring in β, then there is a $receive\_msg^{t,r}(m)$ event in β corresponding to π.

Note that in combination with (DL1), (DL3) implies that there is exactly one $receive\_msg^{t,r}(m)$ event corresponding to each $send\_msg^{t,r}(m)$ event. Now we can define the schedule module $DL^{t,r,M}$. We have already defined $sig(DL^{t,r,M})$. Let $scheds(DL^{t,r,M})$ be the set of sequences β of data link layer actions for which there exists a correspondence relation such that (DL1), (DL2), and (DL3) are all satisfied for β and the correspondence relation.

Although the schedule module $DL^{t,r,M}$ represents the behavior one would require from an interesting data link layer, it is useful for us to define another schedule module $WDL^{t,r,M}$ representing

weaker requirements on data link behavior, in which the layer is required to eventually deliver each message that is sent, but not necessarily in FIFO order. Thus, let $sig(WDL^{t,r,M}) = sig(DL^{t,r,M})$, and let $scheds(WDL^{t,r,M})$ be the set of sequences β of data link layer actions for which there exists a correspondence relation such that (DL1) and (DL3) are satisfied for β and the correspondence.

Although this weaker specification is less interesting than $DL^{t,r,M}$ for describing properties of a useful data link layer, it is adequate for proving our impossibility result. It is easy to see that $WDL^{t,r,M}$ is a weaker specification than $DL^{t,r,M}$, i.e., that $scheds(DL^{t,r,M}) \subseteq scheds(WDL^{t,r,M})$. Thus, any automaton that solves $DL^{t,r,M}$ also solves $scheds(WDL^{t,r,M})$, so that the impossibility results we obtain for solving $WDL^{t,r,M}$ immediately imply corresponding impossibility results for solving $DL^{t,r,M}$.

We have the following immediate consequence of the definition:

Lemma 3.1 *If β is in $scheds(WDL^{t,r,M})$ then the number of $send\_msg^{t,r}$ events in β is equal to the number of $receive\_msg^{t,r}$ events in β.*

4 Data Link Implementation

In this section, we define a "data link protocol", which is intended to be used to implement the data link layer using the services provided by the physical layer. A data link protocol consists of two automata, one at the transmitting station and one at the receiving station. These automata communicate with each other using two physical channels, one in each direction. They also communicate with the outside world, through the data link layer actions we defined in Section 3.

Figure 3 shows how two protocol automata and two physical channels should be connected, in a data link implementation.

4.1 Data Link Protocols

Let t and r again be names (for the transmitting and receiving station respectively). Let M, P_1 and P_2 be alphabets (of messages, forward packets and backwards packets, respectively). Then a *transmitting automaton* for (t,r) and (M,P_1,P_2) is any I/O automaton having the following external action signature.

Input actions:
$send\_msg^{t,r}(m)$, $m \in M$
$receive\_pkt^{r,t}(p)$, $p \in P_2$
Output actions:
$send\_pkt^{t,r}(p)$, $p \in P_1$

In addition, there can be any number of internal actions. That is, a transmitting automaton receives requests from the environment of the data link layer to send messages to the receiving station r. It sends packets to r over the physical channel to r. It also receives packets over the physical channel from r.

Similarly, a *receiving automaton* for (t,r) and (M,P_1,P_2) is any I/O automaton having the following external signature.

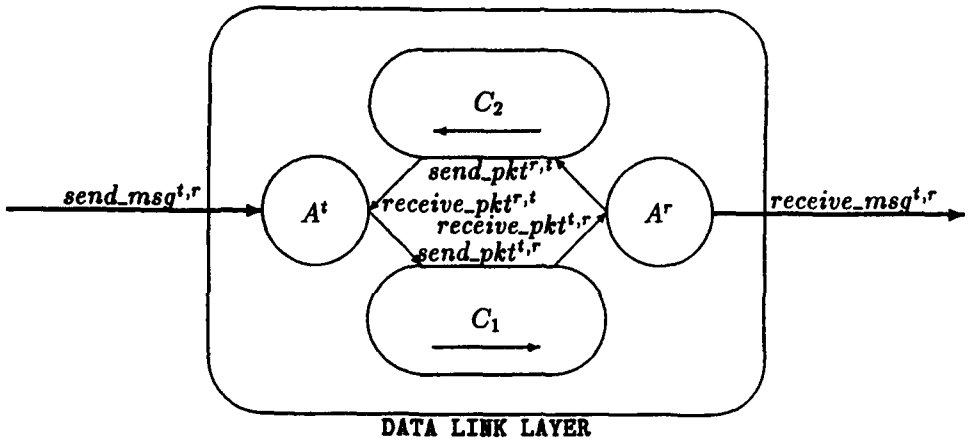

Figure 3: A Data Link Implementation

Input actions:

$receive\_pkt^{t,r}(p)$, $p \in P_1$

Output actions:

$send\_pkt^{r,t}(p)$, $p \in P_2$

$receive\_msg^{t,r}(m)$, $m \in M$

Again, there can also be any number of internal actions. That is, a receiving automaton receives packets over the physical channel from t. It sends packets to t over the physical channel to t, and it delivers messages to the environment of the data link layer.

A *data link protocol* for (t, r) and (M, P_1, P_2) is a pair (A^t, A^r), where A^t is a transmitting automaton for (t, r) and (M, P_1, P_2), and A^r is a receiving automaton for (t, r) and (M, P_1, P_2). (Often we will omit mention of the station names and the alphabets, if these are clear from context.)

4.2 Correctness of Data Link Protocols

Now we are ready to define correctness of data link protocols. Informally, we say that a data link protocol is "correct" provided that when it is composed with any "correct physical layer" (i.e. a pair of FIFO physical channels from t to r and from r to t, respectively), the resulting system yields correct data link layer behavior. This reflects the fundamental idea of layering, that the implementation of one layer should not depend on the details of the implementation of other layers, so that each layer can be implemented and maintained independently. Formally, suppose (A^t, A^r) is a data link protocol for (t, r) and (M, P_1, P_2). We say that (A^t, A^r) is *correct* provided

that the following is true. For all C_1 and C_2 such that C_1 is a FIFO physical channel from t to r with alphabet P_1, and C_2 is a FIFO physical channel from r to t with alphabet P_2, $hide_\Phi(D)$ solves $DL^{t,r,M}$, where D is the composition of A^t, A^r, C_1 and C_2, and Φ is the subset of acts(D) consisting of *send_pkt* and *receive_pkt* actions.

As mentioned earlier, our impossibility results can be proved for weaker data link requirements. Thus we also define *weak correctness* for data link protocols. This is defined exactly as for correctness, except that $hide_\Phi(D)$ is required to solve $WDL^{t,r,M}$ instead of $DL^{t,r,M}$. Obviously, any correct data link protocol is also weakly correct.

Note that the definition of "solves" upon which these correctness definitions are based appears in Appendix A.4. Examination of that definition shows that these correctness definitions require that the fair behaviors of $hide_\Phi(D)$ are all among the schedules of the schedule module $DL^{t,r,M}$ or $WDL^{t,r,M}$.

5 Permissive Physical Channels

Since the correctness of a data link protocol requires that it work when composed with any FIFO physical channels with appropriate alphabets, we are able to prove the impossibility of a correct protocol satisfying certain requirements by merely demonstrating that no such protocol works when combined with a specific pair of FIFO physical channels. In this section we introduce the channels we will use.

5.1 Definitions

We define a physical channel that is parameterized by end stations t and r and a packet alphabet P. It can be considered to be a "very permissive" physical channel. In fact, it can even be considered to be a "universal FIFO physical channel", in the sense of Lemma 5.2 below.

First, we define a set S of ordered pairs (i,j) of positive integers to be a *delivery set* provided that it satisfies the following two conditions: for each positive integer j, S includes a unique element (i,j), and for each positive integer i, it includes at most one element (i,j). We say that a delivery set is *monotone* provided there are no pairs (i_1,j_1) and (i_2,j_2) in S with $i_1 < i_2$ and $j_1 \geq j_2$.

The state of the physical channel $\hat{C}^{t,r,P}$ has two counters, $counter_1$ and $counter_2$, an infinite monotone delivery set S of pairs of positive integers, and a partial mapping *packet* from the set of positive integers to P. The counter $counter_1$ represents the number of *send_pkt* actions, and $counter_2$ represents the number of *receive_pkt* actions, that have occurred so far. The set S determines which packets are delivered, and in what order – it contains pairs (i,j) that correlate the j-th *receive_pkt* event with the i-th *send_pkt* event. Thus the restrictions in the definition of a delivery set correspond to the conditions (DL1). The mapping *packet* associates with an integer i the packet that was sent in the i-th *send_pkt* event. Initially $counter_1$ and $counter_2$ are zero and *packet* is undefined everywhere. In a particular execution, the set S is initialized to an arbitrary monotone delivery set and remains fixed.

The transition relation for the automaton $\hat{C}^{t,r,P}$ consists of all triples (s',π,s) described by the following code.[1]

[1]This style of describing I/O Automata by giving preconditions (that is, conditions on s') and effects (that is,

$send\_pkt^{t,r}(p)$
Effect: $counter_1 \leftarrow counter_1 + 1$
$packet(counter_1) \leftarrow p$

$receive\_pkt^{t,r}(p)$
Precondition: $packet(i) = p$ and $(i, counter_2 + 1) \in S$, for some i
Effect: $counter_2 \leftarrow counter_2 + 1$

The partition puts all the output actions of $\hat{C}^{t,r,P}$ (that is, all the $receive\_pkt^{t,r}(p)$ actions, for all $p \in P$) in a single class.

Lemma 5.1 *The automaton $\hat{C}^{t,r,P}$ is a FIFO physical channel.*

Proof: We must show that $fairbehs(\hat{C}^{t,r,P}) \subseteq scheds(PL - FIFO^{t,r,P})$. Let β be a fair behavior of $\hat{C}^{t,r,P}$. We must show that there exists a correspondence relation that makes (PL1), (PL2) and (PL3) true. Since β is a fair behavior of $\hat{C}^{t,r,P}$, there is a fair execution α of $\hat{C}^{t,r,P}$ with β as its behavior. Let S_0 be the value of the monotone delivery set S in the execution α. We construct a correspondence relation from S_0 as follows: if π is the i-th $send\_pkt^{t,r}$ event in β, and ϕ is the j-th $receive\_pkt^{t,r}$ event in β, then let π correspond to ϕ exactly if $(i,j) \in S_0$. The fact that (PL1) and (PL2) hold for this choice of correspondence relation follows from the properties of monotone delivery sets.

To prove that (PL3) holds, we suppose that on the contrary β contains a finite number N of $receive\_pkt^{t,r}$ events and infinitely many $send\_pkt^{t,r}$ events. Thus in α all the states after the last $receive\_pkt^{t,r}$ event have $counter_2 = N$. Now since S_0 is a monotone delivery set, S_0 contains $(i, N+1)$ for some positive integer i. Let the i-th $send\_pkt^{t,r}$ event in β be $send\_pkt^{t,r}(p)$. Then in every state of α after this event $packet(i) = p$. The code for the automaton shows that in every state of α after the later of the last $receive\_pkt^{t,r}$ event and the i-th $send\_pkt^{t,r}$ event, the action $receive\_pkt^{t,r}(p)$ is enabled. This contradicts the assumption that α is a fair execution.

□

The following converse (which is proved by reversing the construction of the previous proof) shows that $\hat{C}^{t,r,P}$ has among its fair behaviors all of the schedules of the specification $PL - FIFO^{t,r,P}$.

Lemma 5.2 *Suppose β is in $scheds(PL - FIFO^{t,r,P})$. Then $\beta \in fairbehs(\hat{C}^{t,r,P})$.*

We can combine the permissive physical channels with an arbitrary data link protocol, as follows. If A is a data link protocol for (t,r) and (M, P_1, P_2), then let $\hat{D}(A)$ be the composition of A^t, A^r, $\hat{C}^{t,r,P_1}$ and $\hat{C}^{r,t,P_2}$. Also let $\hat{D}'(A) = hide_\Phi(\hat{D}(A))$, where Φ is the subset of $acts(\hat{D}(A))$ consisting of *send_pkt* and *receive_pkt* actions. By virtue of Lemmas 5.1 and 5.2, we have the following result:

Proposition 5.3 *A data link protocol A is correct (respectively, weakly correct) if and only if $\hat{D}'(A)$ solves the specification module $DL^{t,r,M}$ (respectively, $WDL^{t,r,M}$).*

imperatives to be executed sequentially to transform s' to give s) is used in [10]. It is not fundamental to the model, but is rather a notational convenience for describing sets of triples.

The following corollary will be used in our subsequent proofs.

Corollary 5.4 *Suppose that A is a wes a weakly correct data link protocol. Then in every fair behavior of $\hat{D}'(A)$, the number of $send\_msg^{t,r}$ events is equal to the number of $receive\_msg^{t,r}$ events.*

Proof: By Proposition 5.3, it must be that $\hat{D}'(A)$ solves the specification module $WDL^{t,r,M}$, i.e., every fair behavior of $\hat{D}'(A)$ is a schedule of $WDL^{t,r,M}$. The conclusion follows from Lemma 3.1. □

6 Impossibility of Having All Packets Identical

We show here the impossibility of constructing a weakly correct data link protocol that uses only a single type of packet (and so needs no header) to transmit a sequence of identical messages. This result seems weaker than the result we want (that it is impossible to construct a weakly correct data link protocol where all packets contain the same header) but in the next section we will show that in fact the desired result follows from this. By making this simplification we postpone some of the difficult modeling issues, and allow the reader to see the style of impossibility proof in a simpler setting. The technique, of measuring the number of headers by the size of the packet alphabet when the message alphabet has size one, was used earlier without formal proof of a reduction result in [12].

The proof takes the following form: we assume that A is a weakly correct data link protocol in which, in each direction, all packets are identical. Then Corollary 5.4 implies that, in every fair behavior of $\hat{D}'(A)$, the number of $send\_msg^{t,r}$ events is equal to the number of $receive\_msg^{t,r}$ events. We deduce a series of facts about the states of the end stations during executions of $\hat{D}'(A)$, by showing that the failure of one of these facts, coupled with the previously derived facts, would enable us to construct a fair behavior in which the number of $send\_msg^{t,r}$ events is unequal to the number of $receive\_msg^{t,r}$ events. Finally, we use these facts to construct two fair executions of $\hat{D}'(A)$ with identical projections at the receiving automaton, but in one of which two messages are sent while in the other only one message is sent.[2] Since the projections at the receiver are equal, the two executions contain the same number of $receive\_msg^{t,r}$ events. Thus one of them will have the number of $send\_msg^{t,r}$ events unequal to the number of $receive\_msg^{t,r}$ events. This yields a contradiction to the original assumption that the protocol was weakly correct.

Now, we will restrict our attention to the situation where each of A^t and A^r is *deterministic*, that is, at most one locally controlled action is enabled in each state, and at most one new state can be reached by applying an action in a state. As we will see later, this involves no loss of generality. Given a state of an end station (i.e., A^t or A^r) in such a protocol, there is a *unique* maximal execution fragment of that automaton that commences with the given state and includes no input actions. (This execution corresponds to running the automaton from the given state in such a way that it receives no inputs, for as long as it can keep taking steps.) We will say that the given state is *quiescing* if this fragment contains only finitely many $send\_pkt^{t,r}$ (or $send\_pkt^{r,t}$, as appropriate) events.

[2] That is, in these situations the receiver does not know how many messages were sent.

Our first lemma says that from any state in any execution, if the transmitting automaton is run without receiving any inputs (that is, with no $send\_msg^{t,r}$ or $receive\_pkt^{t,r}$ events) then it must send only finitely many packets to the receiver.

Lemma 6.1 *If A is a weakly correct data link protocol for (t,r) and (M,P_1,P_2) with $|P_1| = |P_2| = 1$, A^t and A^r are deterministic, and α is a finite execution of $\hat{D}'(A)$, then A^t is quiescing in the final state of α.*

Proof: The idea of the proof is as follows. If A^t sends infinitely many packets with no response then A^r has no hope of determining how many $send\_msg^{t,r}$ events have happened. In particular, we show that A^r cannot tell the difference between the situation in which one additional $send\_msg^{t,r}$ event occurs after the given finite execution and the situation in which no such events occur.

More precisely, suppose that A^t is not quiescing in the final state of α. Let $\beta = sched(\alpha)$. Then consider the schedule $\beta send\_msg^{t,r}(m')$ where m' is some arbitrary message in the message alphabet of A. This schedule has an extension that is fair and contains no extra $send\_msg^{t,r}$ events (by Lemma A.1). By Corollary 5.4, there is a finite prefix of this extension, say $\beta send\_msg^{t,r}(m')\beta'$ which contains as many $receive\_msg^{t,r}$ events as there are $send\_msg^{t,r}$ events in $\beta send\_msg^{t,r}(m')$, namely, one more than the number of $send\_msg^{t,r}$ events in α.

Let k be the number of $receive\_pkt^{t,r}$ events in β'. Since we assumed that α ended with A^t not quiescing, we can find a schedule of A^t that is an extension of $\beta|A^t$, say $(\beta|A^t)\gamma$, with γ containing only output actions of A^t, including k $send\_pkt^{t,r}$ events (but no $send\_msg^{t,r}$ or $receive\_pkt^{r,t}$ events).

Now we consider the sequence $\beta\gamma(\beta'|A^r)$ of actions of $\hat{D}'(A)$. We show that this sequence is a (not necessarily fair) schedule of $\hat{D}'(A)$, by showing that its projection on each of the four components of the system is a schedule of that component.

1. This sequence has projection on A^t equal to $(\beta|A^t)\gamma$, which is a schedule of A^t by construction.

2. It has projection on A^r equal to $(\beta\beta')|A^r$ (since γ involves only actions of A^t), and this is a schedule of A^r since it is equal to $(\beta send\_msg^{t,r}(m')\beta')|A^r$.

3. It has projection on $\hat{C}^{t,r,P_1}$ equal to $(\beta|\hat{C}^{t,r,P_1})(\gamma|\hat{C}^{t,r,P_1})((\beta'|A^r)|\hat{C}^{t,r,P_1})$, which is a schedule of $\hat{C}^{t,r,P_1}$ since $\gamma|\hat{C}^{t,r,P_1}$ is a sequence of k $send\_pkt^{t,r}(p)$ events, and $(\beta'|A^r)|\hat{C}^{t,r,P_1}$ is a sequence of k $receive\_pkt^{t,r}(p)$ events, where p is the unique element of the packet alphabet P_1.

4. It has projection on $\hat{C}^{r,t,P_2}$ equal to $(\beta|\hat{C}^{r,t,P_2})((\beta'|A^r)|\hat{C}^{r,t,P_2})$ (since γ involves only output actions of A^t, and thus no actions of $\hat{C}^{r,t,P_2}$) and this is a schedule of $\hat{C}^{r,t,P_2}$ since $(\beta'|A^r)|\hat{C}^{r,t,P_2}$ consists only of $send\_pkt^{r,t}$ events, which are inputs to $\hat{C}^{r,t,P_2}$ (and I/O automata must be input-enabled).

Since its projection on each component is a schedule of that component, the sequence $\beta\gamma(\beta'|A^r)$ is a schedule of $\hat{D}'(A)$, by Lemma A.4.

Now consider the two schedules $\beta\gamma(\beta'|A^r)$ and $(\beta)send\_msg^{t,r}(m')\beta'$. They both have the same projection on A^r and hence contain the same number of $receive\_msg^{t,r}$ events. By the argument at the beginning of this proof, this number is exactly one more than the number of $send\_msg^{t,r}$

events in β. On the other hand, the number of $send\_msg^{t,r}$ events in $\beta\gamma(\beta'|A^r)$ is the same as the number of $send\_msg^{t,r}$ events in $\beta\gamma$ (the two schedules having the same projection on A^t), and this is the same as the number of $send\_msg^{t,r}$ events in β, which we just showed is one fewer than the number of $receive\_msg^{t,r}$ events in same schedule $\beta\gamma(\beta'|A^r)$.

Now when we consider a fair extension of $\beta\gamma(\beta'|A^r)$ that contains no further $send\_msg^{t,r}$ events, we find that it contains more $receive\_msg^{t,r}$ events than $send\_msg^{t,r}$ events. This contradicts Corollary 5.4. Thus our assumption (that the lemma was false) must be invalid. □

Our second lemma says that from any state in any execution, if the receiving automaton is run without receiving any inputs, then it must send infinitely many packets to the transmitter.

Lemma 6.2 *If A is a weakly correct data link protocol for (t,r) and (M,P_1,P_2) with $|P_1| = |P_2| = 1$, A^t and A^r are deterministic, and α is a finite execution of $\hat{D}'(A)$, then A^r is not quiescing in the final state of α.*

Proof: Suppose the contrary: that A^r is quiescing in the final state. Once again, we reach a contradiction by constructing two fair schedules of $\hat{D}'(A)$ with the same number of $receive\_msg^{t,r}$ events, but different numbers of $send\_msg^{t,r}$ events. Let α_1 be the maximal execution fragment of A^t containing no inputs and starting from the state of A^t at the end of α. Similarly let α_2 be the maximal execution fragment of A^r containing no inputs and starting from the state of A^r at the end of α. By definition, the projection of $\alpha\alpha_1$ on A^t is a fair execution of A^t, and likewise the projection of $\alpha\alpha_2$ on A^r is a fair execution of A^r. By Lemma 6.1 α_1 contains only finitely many $send\_pkt^{t,r}$ events, and by assumption α_2 contains only finitely many $send\_pkt^{r,t}$ events. Now we consider any sequence of actions γ formed by interleaving the sequences $sched(\alpha_1)$ and $sched(\alpha_2)$.

We claim that $sched(\alpha)\gamma$ is a fair schedule of $\hat{D}'(A)$. We show this by showing that its projection on each of the four components of the system is a fair schedule of that component.

1. Its projection on A^t is just $(sched(\alpha|A^t))sched(\alpha_1)$, which is a fair schedule of A^t by the definition of α_1.

2. Its projection on A^r is $(sched(\alpha|A^r))sched(\alpha_2)$ which is a fair schedule of A^r.

3. Its projection on $\hat{C}^{t,r,P_1}$ is just the projection of sched(α) on that channel followed by a finite number of $send\_pkt^{t,r}$ events. This is of course a schedule of $\hat{C}^{t,r,P_1}$, and is fair because the delivery set could specify the loss of all of the finite number of packets sent but not delivered.

4. Similarly the projection on $\hat{C}^{r,t,P_2}$ is a fair schedule of $\hat{C}^{r,t,P_2}$.

Since its projection on each component is a fair schedule of that component, Lemma A.4 implies that $sched(\alpha)\gamma$ is a fair schedule of $\hat{D}'(A)$. By Corollary 5.4 the number of $receive\_msg^{t,r}$ events in $sched(\alpha)\gamma$ equals the number of $send\_msg^{t,r}$ events.

Now we construct another fair schedule, $sched(\alpha)send\_msg^{t,r}(m)\gamma'$, which contains the same number of $receive\_msg^{t,r}$ events in $sched(\alpha)\gamma$ but one fewer $send\_msg^{t,r}$ event, which yields a contradiction. More specifically, consider $\alpha send\_msg^{t,r}(m)$, where m is an arbitrary element of the message alphabet.

Let α_3 be the maximal execution fragment of A^t containing no inputs and starting from the state of A^t at the end of $\alpha send\_msg^{t,r}(m)$. By Lemma 6.1, α_3 contains only finitely many $send\_pkt^{t,r}$

events. Now we consider the sequence of actions γ' formed by interleaving (in any fashion) the sequences $sched(\alpha_3)$ and $sched(\alpha_2)$. Just as above, $sched(\alpha)send\_msg^{t,r}(m)\gamma'$ is a fair schedule of $\hat{D}'(A)$, and so by Corollary 5.4, the number of $receive\_msg^{t,r}$ events in $sched(\alpha)send\_msg^{t,r}(m)\gamma'$ is equal to the number of $send\_msg^{t,r}$ events. However, since $sched(\alpha)send\_msg^{t,r}(m)\gamma'|A^r$ and $sched(\alpha)\gamma|A^r$ are both equal to $sched(\alpha|A^r)sched(\alpha_2)$, we see that the number of $receive\_msg^{t,r}$ events in $sched(\alpha)send\_msg^{t,r}(m)\gamma'$ is the same as the number of $receive\_msg^{t,r}$ events in $sched(\alpha)\gamma$. By the equalities proved above, this implies that $sched(\alpha)\gamma$ and $sched(\alpha)send\_msg^{t,r}(m)\gamma'$ contain the same number of $send\_msg^{t,r}$ events, which is false. Thus our assumption (that the lemma was false) is invalid. □

The next lemma further characterizes the behavior of a weakly correct data link protocol by showing that the transmitter must both send and receive infinitely many packets.

Lemma 6.3 *If A is a weakly correct data link protocol for (t,r) and (M,P_1,P_2) with $|P_1| = |P_2| = 1$, A^t and A^r are deterministic, and α is a fair execution of $\hat{D}'(A)$ that contains a finite non-zero number of $send\_msg^{t,r}$ actions, then $\alpha|A^t$ contains infinitely many $send\_pkt^{t,r}$ actions and infinitely many $receive\_pkt^{r,t}$ actions.*

Proof: We show that every other possibility leads to a contradiction.

1. Suppose $\alpha|A^t$ contains infinitely many $send\_pkt^{t,r}$ actions and finitely many $receive\_pkt^{r,t}$ actions. Then there is a suffix of $\alpha|A^t$ that contains no input actions (neither $send\_msg^{t,r}$ nor $receive\_pkt^{r,t}$ actions) but contains infinitely many $send\_pkt^{t,r}$ actions. The state of A^t at the start of this suffix must be not quiescing, which contradicts Lemma 6.1.

2. Suppose $\alpha|A^t$ contains finitely many $send\_pkt^{t,r}$ actions and finitely many $receive\_pkt^{r,t}$ actions. Then $\alpha|A^r$ contains finitely many $receive\_pkt^{t,r}$ actions (since the channel $\hat{C}^{t,r,P_1}$ delivers at most as many packets as were sent) and contains finitely many $send\_pkt^{r,t}$ actions (since a fair execution of $\hat{C}^{r,t,P_2}$ would contain an infinite number of $receive\_pkt^{r,t}$ events if it contained an infinite number of $send\_pkt^{r,t}$ events). Thus there is a suffix of $\alpha|A^r$ that contains no input events and only a finite number of $send\_pkt^{r,t}$ events. The state of A^r at the start of this suffix is quiescing, which contradicts Lemma 6.2.

3. Suppose $\alpha|A^t$ contains finitely many $send\_pkt^{t,r}$ actions and infinitely many $receive\_pkt^{r,t}$ actions. First consider the maximal execution of A^r starting from the initial state of A^r and containing no input actions. This execution is a fair execution of A^r. Let β be the schedule of this execution. By Lemma 6.2, β contains infinitely many $send\_pkt^{r,t}$ actions. Let γ be the sequence of actions obtained by interleaving β and $sched(\alpha|A^t)$ in such a way that for each i the i-th $receive\_pkt^{r,t}$ action is immediately preceded by the i-th $send\_pkt^{r,t}$ action. We claim that γ is a fair schedule of $\hat{D}'(A)$. Its projections on A^t and A^r are fair schedules by construction. Its projection on $\hat{C}^{r,t,P_2}$ is just $send\_pkt^{r,t}(p)receive\_pkt^{r,t}(p)send\_pkt^{r,t}(p)receive\_pkt^{r,t}(p)\ldots$ (where $P_2 = \{p\}$) which is a fair schedule, and its projection on $\hat{C}^{t,r,P_1}$ is a fair schedule since it consists of the sending of a finite number of packets and the delivery of none (as β contains no inputs to A^r, in particular no $receive\_pkt^{t,r}$ actions) and this can be achieved by a suitable delivery set.

We observe that β (and hence γ) cannot contain any $receive\_msg^{t,r}$ action. (Otherwise, take the prefix of β up to and including the first $receive\_msg^{t,r}$ event, regard it as a schedule of $\hat{D}'(A)$ where all the actions take place at A^r, and extend it to a fair schedule of $\hat{D}'(A)$ which contains no inputs to $\hat{D}'(A)$, that is, no $send\_msg^{t,r}$ actions. This contradicts Corollary 5.4.) However, γ contains a non-zero number of $send\_msg^{t,r}$ events (the same ones as in α). Thus γ is a fair schedule of $\hat{D}'(A)$ which does not contain the same number of $receive\_msg^{t,r}$ events as of $send\_msg^{t,r}$ events, contradicting Corollary 5.4.

□

The final lemma allows us to construct two executions that look identical to the receiver, but have different numbers of messages sent at the transmitting end.

Lemma 6.4 *Let A be a weakly correct data link protocol for (t,r) and (M,P_1,P_2) with $|P_1| = |P_2| = 1$, such that A^t and A^r are deterministic. Let α_1 and α_2 be two fair executions of $\hat{D}'(A)$ such that $sched(\alpha_1)$ begins with $send\_msg^{t,r}(m)$ and contains no other $send\_msg^{t,r}$ event, and $sched(\alpha_2)$ begins with $send\_msg^{t,r}(m)send\_msg^{t,r}(m)$ and contains no other $send\_msg^{t,r}$ event. There exist fair executions $\hat{\alpha}_1$ and $\hat{\alpha}_2$ of $\hat{D}'(A)$ such that $\hat{\alpha}_i|A^t = \alpha_i|A^t$ for $i = 1,2$, and such that $\hat{\alpha}_1|A^r = \hat{\alpha}_2|A^r$.*

Proof: Applying Lemma 6.3 to the execution α_1, we see that we may express $sched(\alpha_1|A^t)$ as

$$send\_msg^{t,r}(m)\beta_1^1\gamma_1^1\beta_1^2\gamma_1^2\beta_1^3\ldots$$

where β_1^i consists only of internal or $send\_pkt^{t,r}$ actions, γ_1^i consists only of internal or $receive\_pkt^{r,t}$ actions, and where each β_1^i (except possibly β_1^1) and each γ_1^i contains a finite, non-zero number of $send\_pkt^{t,r}$ or $receive\_pkt^{r,t}$ events.[3] Let a_1^i denote the number of $receive\_pkt^{r,t}$ events in γ_1^i.

Similarly, we see that we may express $sched(\alpha_2|A^t)$ as

$$send\_msg^{t,r}(m)send\_msg^{t,r}(m)\beta_2^1\gamma_2^1\beta_2^2\gamma_2^2\beta_2^3\ldots$$

where β_2^i consists only of internal or $send\_pkt^{t,r}$ actions, γ_2^i consists only of internal or $receive\_pkt^{r,t}$ actions, and where each β_2^i (except possibly β_2^1) and each γ_2^i contains a finite, non-zero number of $send\_pkt^{t,r}$ or $receive\_pkt^{r,t}$ events. Let a_2^i denote the number of $receive\_pkt^{r,t}$ events in γ_2^i.

We construct inductively finite schedules δ_1^j and δ_2^j of $\hat{D}'(A)$ such that

$$\delta_1^j|A^t = send\_msg^{t,r}(m)\beta_1^1\gamma_1^1\ldots\gamma_1^{j-1}\beta_1^j,$$

$$\delta_2^j|A^t = send\_msg^{t,r}(m)send\_msg^{t,r}(m)\beta_2^1\gamma_2^1\ldots\gamma_2^{j-1}\beta_2^j,$$

$\delta_1^j|A^r = \delta_2^j|A^r$, and δ_i^j is an extension of δ_i^{j-1} for $i = 1,2$.

The base case of the construction is straightforward, as we put $\delta_1^1 = send\_msg^{t,r}(m)\beta_1^1$ and $\delta_2^1 = send\_msg^{t,r}(m)send\_msg^{t,r}(m)\beta_2^1$. Suppose δ_1^{j-1} and δ_2^{j-1} have been constructed. By Lemma 6.2, the (uniquely defined, by determinism) state of A^r at the end of δ_1^{j-1} is not quiescing, and therefore there is an execution fragment of A^r starting from that state containing $\max(a_1^{j-1}, a_2^{j-1})$

[3]The exception is due to the fact that we do not know whether the first packet sent by A^t precedes or follows the first packet received by A^t.

$send\_pkt^{r,t}$ events and (possibly) internal events and $receive\_msg^{t,r}$ events, but no input events, (i.e., no $receive\_pkt^{t,r}$ events). Let the schedule of this execution fragment be η^j. We set $\delta_1^j = \delta_1^{j-1}\eta^j\gamma_1^{j-1}\beta_1^j receive\_pkt^{t,r}(p)$, where $P_1 = \{p\}$. We show that δ_1^j is a schedule of $\hat{D}'(A)$ by showing that its projection at each component is a schedule of that component:

1. The projection on A^t is $(\delta_1^{j-1}|A^t)\gamma_1^{j-1}\beta_1^j$, which is a prefix of $sched(\alpha_1)|A^t$ and so is a schedule of A^t.

2. The projection on A^r is $(\delta_1^{j-1}|A^r)\eta^j receive\_pkt^{t,r}(p)$, which is a schedule of A^r because η^j can occur starting from the state after $\delta_1^{j-1}|A^r$ and $receive\_pkt^{t,r}$ is an input action of A^r.

3. Its projection on $\hat{C}^{t,r,P_1}$ is a schedule of $\hat{C}^{t,r,P_1}$ because β_1^j contains at least one $send\_pkt^{t,r}(p)$ event. (The rest may be "lost".)

4. Its projection on $\hat{C}^{t,r,P_2}$ is a schedule of $\hat{C}^{t,r,P_2}$ because η^j contains at least a_1^{j-1} $send\_pkt^{r,t}(p')$ actions, so that there are enough packets sent in δ_1^j to account for those received.

We now set $\delta_2^j = \delta_2^{j-1}\eta^j\gamma_2^{j-1}\beta_2^j receive\_pkt^{t,r}(p)$. The same argument as for δ_1^j shows that δ_2^j is a schedule of $\hat{D}'(A)$. The fact that $\delta_1^j|A^r = \delta_2^j|A^r$ follows easily using the inductive hypothesis that $\delta_1^{j-1}|A^r = \delta_2^{j-1}|A^r$.

Thus, it is clear that δ_1^j and δ_2^j have the properties claimed for them. Now let δ_1 denote the limit of the successively extended schedules δ_1^j. Similarly let δ_2 denote the limit of δ_2^j. Thus δ_i is a schedule of $\hat{D}'(A)$, for $i = 1, 2$. We claim that in fact δ_i is a fair schedule of $\hat{D}'(A)$. To show this we consider the projection on each component. Each of the projections $\delta_i|A^t$ and $\delta_i|A^r$ contains infinitely many locally controlled events by construction, and so is fair (using determinism). Each of the projections on a physical channel contains infinitely many packet deliveries, and so is fair by the definition of the permissive channels. Let $\hat{\alpha}_i$ be a fair execution of $\hat{D}'(A)$ with schedule δ_i for $i = 1, 2$. This completes the construction. □

Proposition 6.5 *There is no weakly correct data link protocol for (t, r) and (M, P_1, P_2) with $|P_1| = |P_2| = 1$.*

Proof: Assume that A is a weakly correct data link protocol with $|P_1| = |P_2| = 1$.

First, we deal with the potential non-determinism of the end-stations. By a result of Goldman and Lynch [7], there is a deterministic automaton B^t (respectively, B^r) with fair behaviors that are a subset of the fair behaviors of A^t (respectively, A^r).[4] Put $B = (B^t, B^r)$, which is also a data link protocol with $|P_1| = |P_2| = 1$. Now by Lemmas A.2 and A.4, $fairbehs(\hat{D}'(B)) \subseteq fairbehs(\hat{D}'(A))$, and so B is weakly correct (using Lemma 5.3 and the fact that A is weakly correct).

Now, let m be an arbitrary element of the message alphabet. By Lemma A.1, there are fair executions α_1 and α_2 of $\hat{D}'(B)$ such that $sched(\alpha_1)$ begins with $send\_msg^{t,r}(m)$ and contains no other $send\_msg^{t,r}$ event, and $sched(\alpha_2)$ begins with $send\_msg^{t,r}(m)send\_msg^{t,r}(m)$ and contains no other $send\_msg^{t,r}$ event. Consider the fair executions $\hat{\alpha}_1$ and $\hat{\alpha}_2$ whose existence is shown in Lemma 6.4. Since the protocol B is weakly correct, each $\hat{\alpha}_i$ satisfies Corollary 5.4 (that is,

[4]The proof is actually left as an exercise in [7]. However, the idea is not hard: given an I/O automaton that is not necessarily deterministic, one first removes all except one step (s', π, s) for each given s' and partition class. Then one simulates the resulting machines with a single-class machine that gives fair turns to all the classes.

the number of $receive\_msg^{t,r}$ events in $\hat{\alpha}_i$ equals the number of $send\_msg^{t,r}$ events in $\hat{\alpha}_i$). Now the number of $send\_msg^{t,r}$ events in $\hat{\alpha}_i$ is just the number of $send\_msg^{t,r}$ events in $\hat{\alpha}_i|B^t$ which equals the number of $send\_msg^{t,r}$ events in $\alpha_i|B^t$ by the properties of $\hat{\alpha}_i$. Since $\alpha_i|B^t$ contains i $send\_msg^{t,r}$ events, we deduce that $\hat{\alpha}_i$ contains i $receive\_msg^{t,r}$ events, contradicting the fact that $\hat{\alpha}_1|B^r$ and $\hat{\alpha}_2|B^r$ are equal, and so contain the same number of $receive\_msg^{t,r}$ events. □

7 Defining Headers in Protocols with Infinite Packet Alphabet

Most data link protocols in the literature use a finite packet alphabet in each direction, since packets are required to be of limited size. However, it is normally the case that the packets are treated as having two separate parts: a header (which determines what is to be done with the packet) and an encapsulated message (treated as an uninterpreted bit string). Indeed, one can envisage protocols that allow packets of unbounded size because the included messages may have unbounded size, and yet use only a fixed size of header (and thus a finite number of headers). Here we sketch one way in which one can model the existence of headers in a protocol, without assuming that the packets are necessarily structured explicitly with two parts, one a control field and the other an uninterpreted message.

We model the "headers" used by a protocol as follows. Let $A = (A^t, A^r)$ be a data link protocol for (t, r) and (M, P_1, P_2). Let $\equiv$ be an equivalence relation on the domain $M \cup P_1 \cup P_2 \cup states(A^t) \cup states(A^r) \cup acts(A^t) \cup acts(A^r)$. Then $\equiv$ is said to be a *header relation* for A provided that the following conditions hold.

1. $\equiv$ only relates elements of the same kind, i.e., elements of M, or P_1, or $states(A^t)$, etc. Also, a start state cannot be related to a non-start state. Moreover, if $a \equiv a'$ for two actions a and a', then a and a' are identical except possibly for a difference in their message or packet parameter. Further, every pair a and a' of locally controlled events of A^t (respectively, of A^r) such that $a \equiv a'$, a and a' are in the same class of $part(A^t)$ (respectively, of $part(A^r)$).
2. For each pair m, m' of messages in M, $send\_msg^{t,r}(m) \equiv send\_msg^{t,r}(m')$ if and only if $m \equiv m'$, and $receive\_msg^{t,r}(m) \equiv receive\_msg^{t,r}(m')$ if and only if $m \equiv m'$.
3. For each pair p,p' of packets in P_1, $send\_pkt^{t,r}(p) \equiv send\_pkt^{t,r}(p')$ if and only if $p \equiv p'$, and $receive\_pkt^{t,r}(p) \equiv receive\_pkt^{t,r}(p')$ if and only if $p \equiv p'$.
4. For each pair p,p' of packets in P_2, $send\_pkt^{r,t}(p) \equiv send\_pkt^{r,t}(p')$ if and only if $p \equiv p'$, and $receive\_pkt^{r,t}(p) \equiv receive\_pkt^{r,t}(p')$ if and only if $p \equiv p'$.
5. For every two states q and q' of A^t (respectively, of A^r) with $q \equiv q'$, if action a is enabled in q then there is an action a' with $a \equiv a'$, such that a' is enabled in q'.
6. For every two states q and q' of A^t (respectively, of A^r) and every two actions a and a' of A^t (respectively, of A^r) such that $q \equiv q'$ and $a \equiv a'$, if r is a state such that (q, a, r) is a step of A^t (respectively, of A^r) and action a' is enabled in state q', then there exists a state r' such that $r \equiv r'$ and (q', a', r') is a step of A^t (respectively, of A^r).

For a data link protocol A for (t, r) and (M, P_1, P_2) with a header relation $\equiv$, we define the set $headers(A, t, r, \equiv)$ to be the set of equivalence classes of packets in P_1. Similarly $headers(A, r, t, \equiv)$ is the set of equivalence classes of packets in P_2. We think of each equivalence class of packets as

being those (in one direction) with the same pattern of bits in the header. Informally, the way a packet is processed must depend only on the header – for example, if receiving a packet takes the protocol to a state where release of a message to the higher layer is possible, then receiving any other packet containing the same header will also take the protocol to a state where release of a message to the higher layer is possible (however, it may be a different message that is released!) We note that for a data link protocol A, the diagonal relation, where each message, action etc. is equivalent only to itself, is a header relation for A. We say that A has *no header under* $\equiv$ if each of $headers(A, t, r, \equiv)$ and $headers(A, r, t, \equiv)$ is a singleton set, that is, all packets in P_1 are related by $\equiv$, as are all packets in P_2. We say that A has *no header* if there exists a header relation $\equiv$ for A such that A has no header under $\equiv$.

In order to prove that headers are necessary for a data link protocol, we show how to reduce the question of the existence of a protocol with sets of header equivalence classes of a given size, to the question of the existence of a protocol using packet alphabets of that size. This will allow us to show that there is no weakly correct data link protocol that has no header using our earlier result that there is no weakly correct data link protocol with packet alphabets of size one.

Proposition 7.1 *Suppose $A = (A^t, A^r)$ is a weakly correct data link protocol for (t, r) and (M, P_1, P_2). If $\equiv$ is a header relation for A such that $|headers(A, t, r, \equiv)| = h_1$ and $|headers(A, r, t, \equiv)| = h_2$, then there are alphabets M', P_1' and P_2' with $|M'| = 1$, $|P_1'| = h_1$ and $|P_2'| = h_2$ and a weakly correct data link protocol $B = (B^t, B^r)$ for (t, r) and (M', P_1', P_2').*

Proof: Choose m to be an arbitrary element of M and put $M' = \{m\}$, $P_1' = headers(A, t, r, \equiv)$ and $P_2' = headers(A, r, t, \equiv)$.[5] These alphabets clearly have the correct cardinalities. Now let B^t be the transmitting automaton for (t, r) and (M', P_1', P_2') defined as follows. The input actions of B^t are $send\_msg^{t,r}(m)$ and $receive\_pkt^{r,t}(p')$ where p' is an element of P_2', the output actions are $send\_pkt^{t,r}(p')$ where p' is an element of P_1', and the internal actions are the internal actions of A^t. We say that an action π of A^t *is represented by* an action π' of B^t exactly when one of the following conditions holds:

- π' is either $send\_msg^{t,r}(m)$ or an internal action of A^t and $\pi = \pi'$
- π' is $send\_pkt^{t,r}(p')$ and $\pi = send\_pkt^{t,r}(p)$ for some p which is an element of p'
- π' is $receive\_pkt^{r,t}(p')$ and $\pi = receive\_pkt^{r,t}(p)$ for some p which is an element of p'.

The states and start states of B^t are the same as those of A^t. The transition relation of B^t includes (s', π', s) exactly when there exists some π that is represented by π' for which $(s', \pi, s) \in steps(A^t)$. The partition $part(B^t)$ relates locally controlled actions π_1' and π_2' exactly when $part(A^t)$ relates some (and hence all) pairs π_1 and π_2 such that π_1 is represented by π_1' and π_2 is represented by π_2'.

Similarly let B^r be the receiving automaton for (t, r) and (M', P_1', P_2') defined as follows. The input actions of B^r are $receive\_pkt^{t,r}(p')$ where p' is an element of P_1', the output actions are $receive\_msg^{t,r}(m)$ and $send\_pkt^{r,t}(p')$ where p' is an element of P_2', and the internal actions are the internal actions of A^r. We say that an action π of A^r *is represented by* an action π' of B^r exactly when one of the following conditions holds:

[5] Thus each packet name in the alphabet P_1' is a set of packet names from the alphabet P_1.

- π' is either $receive\_msg^{t,r}(m)$ or an internal action of A^r and $\pi = \pi'$
- π' is $send\_pkt^{r,t}(p')$ and $\pi = send\_pkt^{r,t}(p)$ for some p which is an element of p'
- π' is $receive\_pkt^{t,r}(p')$ and $\pi = receive\_pkt^{t,r}(p)$ for some p which is an element of p'.

The states and start states of B^r are the same as those of A^r. The transition relation of B^r includes (s', π', s) exactly when there exists some π that is represented by π' for which $(s', \pi, s) \in steps(A^r)$. The partition $part(B^r)$ relates locally controlled actions π_1' and π_2' exactly when $part(A^r)$ relates some (and hence all) pairs π_1 and π_2 such that π_1 is represented by π_1' and π_2 is represented by π_2'.

It is easy to check that (B^t, B^r) is a data link protocol for (t, r) and (M', P_1', P_2'). We claim that it is weakly correct, proving the proposition. To prove the claim we will consider an arbitrary fair execution $s_0', \pi_1', s_1', \pi_2', s_2', \ldots$ of $\hat{D}(B)$. From this we can construct an execution $s_0, \pi_1, s_1, \pi_2, s_2, \ldots$ of $\hat{D}(A)$ such that π_i is represented by π_i' for each i, the state of A^t (respectively, of A^r) in s_i is the same as the state of B^t (respectively, of B^r) in s_i', and the state of $\hat{C}^{t,r,P_1}$ (respectively, of $\hat{C}^{r,t,P_2}$) in s_i is related tc the state of $\hat{C}^{t,r,P_1'}$ (respectively, of $\hat{C}^{r,t,P_2'}$) in s_i' in the natural way: the values for the variables S, $counter_1$ and $counter_2$ are the same in $\hat{C}^{t,r,P_1}$ (respectively, in $\hat{C}^{r,t,P_2}$) as in $\hat{C}^{t,r,P_1'}$ (respectively, in $\hat{C}^{r,t,P_2'}$), and for each n the value of $packet(n)$ in $\hat{C}^{t,r,P_1}$ (respectively, in $\hat{C}^{r,t,P_2}$) is one element of its value in $\hat{C}^{t,r,P_1'}$ (respectively, in $\hat{C}^{r,t,P_2'}$) except when both values are undefined.[6] This execution is in fact a fair execution of $\hat{D}(A)$, as is seen by observing that no action $receive\_msg^{t,r}(m')$ for $m' \neq m$ is enabled in any state s_i (using the weak correctness of A and the fact that no action π_j is $send\_msg^{t,r}(m')$), and that therefore if a locally controlled action of $\hat{D}(A)$ is enabled in s_i then it is represented by a locally controlled action of $\hat{D}(B)$ that is enabled in s_i'. Since this execution is fair, its behavior is a schedule of $WDL^{t,r,M}$. However the two executions have identical behavior (the actions differ only for *send_pkt* and *receive_pkt* events, which are hidden). Thus there is a correspondence between $send\_msg^{t,r}$ and $receive\_msg^{t,r}$ events that satisfies (DL1) and (DL3). Thus $\hat{D}(B)$ satisfies $WDL^{t,r,M'}$, and so by Proposition 5.3, B is weakly correct. □

Theorem 7.2 *There is no weakly correct data link protocol that has no header.*

Proof: Immediate from Proposition 6.5 and Proposition 7.1. □

Acknowledgements: We thank Yishay Mansour for his help in our initial discussions about this work, and also Steve Ponzio and Isaac Saias for their useful suggestions for improving the presentation.

References

[1] Attiya, H., Fischer, M., Wang, D.-W., and Zuck, L., "Reliable Communication Using Unreliable Channels", manuscript.

[2] Bartlett, K., Scantlebury, R., and Wilkinson, P., "A Note on Reliable Full-Duplex Transmission over Half-Duplex Links" *Communications of the ACM* 12(5):260-261, May 1969.

[6] The inductive construction of the execution $s_0, \pi_1, s_1, \ldots$ is straightforward.

[3] Chou, C.-T., and Gafni, E., "Understanding and Verifying Distributed Algorithms Using Stratified Decomposition" *Proceedings of 7th ACM Symposium on Principles of Distributed Computing* pp. 44-65, August 1988.

[4] Fekete, A., Lynch, N., and Shrira, L., "A Modular Proof of Correctness for a Network Synchronizer" *Proceedings of the 2nd International Workshop on Distributed Algorithms*, Amsterdam, Netherlands, July 1987, (J. van Leeuwen, ed), pp. 219-256. Lecture Notes in Computer Science 312, Springer-Verlag.

[5] Fekete, A., Lynch, N. A., Mansour, Y. and Spinelli, J., "The Data Link Layer: The Impossibility of Implementation in Face of Crashes", Technical Memo, TM-355b, Laboratory for Computer Science, Massachusetts Institute of Technology.

[6] Fekete, A, Lynch, N., Merritt, M., and Weihl, W., "Commutativity-Based Locking for Nested Transactions" to appear in JCSS.

[7] Lynch, N., and Goldman, K., "Distributed Algorithms" Research Seminar Series MIT/LCS/RSS-5, Laboratory for Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 1989.

[8] Lynch, N. A., Mansour, Y. and Fekete, A., "Data Link Layer: Two Impossibility Results," *Proceedings of 7th ACM Symposium on Principles of Distributed Computing* pp. 149-170, August 1988.

[9] Lynch N. A. and Tuttle M. R., "Hierarchical Correctness Proofs for Distributed Algorithms," *Proceedings of the 6th ACM Symposium on Principles of Distributed Computing* pp. 137-151, August 1987.

[10] Lynch, N., and Tuttle, M., "An Introduction to Input/Output Automata" *CWI Quarterly*, 2(3):219-246, September 1989.

[11] Lynch, N., "A Hundred Impossibility Proofs for Distributed Computing", *Proceedings of 8th ACM Symposium on Principles of Distributed Computing*, pp. 1-28, August 1989.

[12] Mansour, Y., and Schieber, B., "The Intractability of Bounded Protocols for non-FIFO Channels" *Proceedings of 8th ACM Symposium on Principles of Distributed Computing*, pp. 59-72, August 1989.

[13] Stenning, N., "A Data Transfer Protocol" *Computer Networks* 1:99-110, 1976.

[14] Troxel, G., "A Hierarchical Proof of an Algorithm for Deadlock Recovery in a System using Remote Procedure Calls" MS thesis, Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science Cambridge, MA. January, 1990.

[15] Welch, J., Lamport, L., and Lynch, N., "A Lattice-Structured Proof of a Minimum Spanning tree Algorithm" *Proceedings of 7th ACM Symposium on Principles of Distributed Computing*, pp. 28-43, August 1988.

[16] Wang, D.-W., and Zuck, L., "Tight Bounds for the Sequence Transmission Problem", *Proceedings of 8th ACM Symposium on Principles of Distributed Computing*, pp. 73-84, August 1989.

A The I/O Automaton Model

The *input/output automaton* model was defined in [9] as a tool for modeling concurrent and distributed systems. We refer the reader to [9] and to the expository paper [10] for a complete development of the model, plus motivation and examples. Here, we provide a brief summary of those aspects of the model that are needed for our results.

A.1 Actions and Action Signatures

We assume a universal set of *actions*, and we refer to a particular occurrence of an action in a sequence as an *event*.

An *action signature* S is an ordered triple consisting of three pairwise-disjoint sets of actions. We write $in(S)$, $out(S)$ and $int(S)$ for the three components of S, and refer to the actions in the three sets as the *input actions*, *output actions* and *internal actions* of S, respectively. We let $ext(S) = in(S) \cup out(S)$ and refer to the actions in $ext(S)$ as the *external actions* of S. Also, we let $local(S) = out(S) \cup int(S)$, and refer to the actions in $local(S)$ as the *locally-controlled actions* of S. Finally, we let $acts(S) = in(S) \cup out(S) \cup int(S)$, and refer to the actions in $acts(S)$ as the *actions* of S. An *external action signature* is an action signature consisting entirely of external actions, that is, having no internal actions.

A.2 Input/Output Automata

An *input/output automaton* A (also called an *I/O automaton* or simply an *automaton*) consists of five components:

1. an action signature $sig(A)$,
2. a set $states(A)$ of *states*,
3. a nonempty set $start(A) \subseteq states(A)$ of *start states*,
4. a transition relation $steps(A) \subseteq (states(A) \times acts(sig(A)) \times states(A))$, with the property that for every state s' and input action π there is a transition (s', π, s) in $steps(A)$, and
5. an equivalence relation $part(A)$ on $local(sig(A))$, having at most countably many equivalence classes.

We refer to an element (s', π, s) of $steps(A)$ as a *step* of A. The step (s', π, s) is called an *input step* of A if π is an input action. *Output steps*, *internal steps*, *external steps* and *locally-controlled steps* are defined analogously. If (s', π, s) is a step of A, then π is said to be *enabled* in s'. Since every input action is enabled in every state, automata are said to be *input-enabled*. The partition $part(A)$ is an abstract description of the underlying components of the automaton, and is used to define fairness.

An *execution fragment* of A is a finite sequence $s_0\pi_1 s_1 \pi_2 \ldots \pi_n s_n$ or an infinite sequence $s_0\pi_1 s_1 \pi_2 \ldots \pi_n s_n \ldots$ of alternating states and actions of A such that $(s_i, \pi_{i+1}, s_{i+1})$ is a step of A for every i. An execution fragment beginning with a start state is called an *execution*. We denote the set of executions of A by $execs(A)$. A state is said to be *reachable* in A if it is the final state of a finite execution of A.

A *fair execution* of an automaton A is defined to be an execution α of A such that the following condition holds for each class C of $part(A)$: if α is finite, then no action of C is enabled in the final state of α, while if α is infinite, then either α contains infinitely many events from C, or else α contains infinitely many occurrences of states in which no action of C is enabled. Thus, a fair execution gives "fair turns" to each class of $part(A)$. Informally, one class of $part(A)$ typically consists of all the actions that are controlled by a single subsystem within the system modeled by the automaton A, and so fairness means giving each such subsystem regular opportunities to take a step under its control, if any is enabled. In the common case that there is no lower level of structure to the system modeled by A (when all locally-controlled actions are in a single class of $part(A)$), a fair execution is an execution in which infinitely often the automaton is given an opportunity to take a locally controlled action if any is enabled. We denote the set of fair executions of A by $fairexecs(A)$.

The *schedule* of an execution fragment α of A is the subsequence of α consisting of actions, and is denoted by $sched(\alpha)$. We say that β is a *schedule* of A if β is the schedule of an execution of A. We denote the set of schedules of A by $scheds(A)$. We say that β is a *fair schedule* of A if β is the schedule of a fair execution of A and we denote the set of fair schedules of A by $fairscheds(A)$.

The *behavior* of an execution or schedule α of A is the subsequence of α consisting of external actions, and is denoted by $beh(\alpha)$. We say that β is a *behavior* of A if β is the behavior of an execution of A. We denote the set of behaviors of A by $behs(A)$. We say that β is a *fair behavior* of A if β is the behavior of a fair execution of A and we denote the set of fair behaviors of A by $fairbehs(A)$. When an algorithm is modeled as an I/O automaton, it is the set of fair behaviors of the automaton that reflect the activity of the algorithm that is important to users. An important operation on schedules or other sequences is *projection*. If α is a sequence (of elements of any alphabet) and Φ is a set of elements, we write $\alpha|\Phi$ for the subsequence of α consisting of the occurrences of those elements in the set Φ. Thus if α is an execution or schedule of A, then $beh(\alpha) = \alpha|ext(A)$.

We say that a finite behavior or schedule β of A *can leave* A *in state* s if there is a finite execution α with β as its behavior or schedule, such that the final state in α is s.

The following lemma says that no matter what has happened in any finite execution, and no matter what inputs continue to arrive from the environment, an automaton can continue to take steps to give a fair execution.

Lemma A.1 *Let A be an I/O automaton and let γ be a sequence of input actions of A.*

1. *Suppose that α is a finite execution of A. Then there exists a fair execution α' of A such that α' is an extension of α and $beh(\alpha')|in(A) = (beh(\alpha)|in(A))\gamma$.*
2. *Suppose that β is a finite schedule of A. Then there exists a fair schedule β' of A such that β' is an extension of β and $\beta'|in(A) = (\beta|in(A))\gamma$.*

A.3 Schedule Modules

In line with our approach, where the facts about an algorithm that are important to its users are modeled by the set of fair behaviors of an automaton, we also give a formal model for a problem specification by a set of sequences of actions. More precisely, a problem will be specified by a pair consisting of an action signature and a set of sequences over the actions in that signature. (In most interesting cases, the action signature will be an external action signature.) The mathematical object used to describe a problem is called a "schedule module".

A *schedule module* H consists of two components:

1. an action signature $sig(H)$, and
2. a set $scheds(H)$ of *schedules.*

Each schedule in $scheds(H)$ is a finite or infinite sequence of actions of H.

The *behavior* of a schedule β of H is the subsequence of β consisting of external actions, and is denoted by $beh(\beta)$. We say that β is a *behavior* of H if β is the behavior of an execution of H. We denote the set of behaviors of H by $behs(H)$. We extend the definitions of fair schedules and fair behaviors to schedule modules in a trivial way, letting $fairscheds(H) = scheds(H)$ and $fairbehs(H) = behs(H)$.

We use the term *module* to designate either an automaton or schedule module. If M is a module, we sometimes write $acts(M)$ as shorthand for $acts(sig(M))$, and likewise for $in(M)$, $out(M)$, etc. If β is any sequence of actions and M is a module, we write $\beta|M$ for $\beta|acts(M)$.

A.4 Solving Problems

Now we are ready to define our notion of "solving". This notion is intended for describing the way in which particular algorithms (formalized as automata) solve particular problems (formalized as schedule modules). Let A be an automaton and H a schedule module with the same external action signature as A. Then we say that A *solves* H if $fairbehs(A) \subseteq behs(H)$.

A.5 Composition

The most useful way of combining I/O automata is by means of a composition operator, as defined in this subsection. This models the way algorithms interact, as for example when the pieces of a communication protocol at different nodes and a lower-level protocol all work together to provide a higher-level service.

A.5.1 Composition of Action Signatures

Let I be an index set that is at most countable. A collection $\{S_i\}_{i \in I}$ of action signatures is said to be *strongly compatible* if for all $i, j \in I$, we have

1. $out(S_i) \cap out(S_j) = \emptyset$,
2. $int(S_i) \cap acts(S_j) = \emptyset$, and
3. no action is in $acts(S_i)$ for infinitely many i.

Thus, no action is an output of more than one signature in the collection, and internal actions of any signature do not appear in any other signature in the collection.

The *composition* $S = \Pi_{i \in I} S_i$ of a collection of strongly compatible action signatures $\{S_i\}_{i \in I}$ is defined to be the action signature with $in(S) = \cup_{i \in I} in(S_i) \setminus \cup_{i \in I} out(S_i)$, $out(S) = \cup_{i \in I} out(S_i)$, and $int(S) = \cup_{i \in I} int(S_i)$. Thus, output actions are those that are outputs of any of the component signatures, and similarly for internal actions. Input actions are any actions that are inputs to any of the component signatures, but outputs of no component signature.

A.5.2 Composition of Automata

A collection $\{A_i\}_{i \in I}$ of automata is said to be *strongly compatible* if their action signatures are strongly compatible. The *composition* $A = \Pi_{i \in I} A_i$ of a strongly compatible collection of automata $A_{i i \in I}$ has the following components:

1. $sig(A) = \Pi_{i \in I} sig(A_i)$,
2. $states(A) = \Pi_{i \in I} states(A_i)$[7]
3. $start(A) = \Pi_{i \in I} start(A_i)$
4. $steps(A)$ is the set of triples (s_1, π, s_2) such that for all $i \in I$, if $\pi \in acts(A_i)$ then $(s_1[i], \pi, s_2[i]) \in steps(A_i)$, and if $\pi \notin acts(A_i)$ then $s_1[i] = s_2[i]$[8], and
5. $part(A) = \cup_{i \in I} part(A_i)$.

Since the automata A_i are input-enabled, so is their composition, and hence their composition is an automaton. Each step of the composition automaton consists of all the automata that have a particular action in their signatures performing that action concurrently, while the automata that do not have that action in their signatures do nothing. The partition for the composition is formed by taking the union of the partitions for the components. Thus, a fair execution of the composition gives fair turns to all of the classes within all of the component automata. In other words, all component automata in a composition continue to act autonomously. If $\alpha = s_0 \pi_1 s_1 \ldots$ is an execution of A, let $\alpha | A_i$ be the sequence obtained by deleting $\pi_j s_j$ when π_j is not an action of A_i, and replacing the remaining s_j by $s_j[i]$.

The following basic results relate executions, schedules and behaviors of a composition to those of the automata being composed. The first result says that the projections of executions of a composition onto the components are executions of the components, and similarly for schedules, etc. The parts of this result dealing with fairness depend on the fact that at most one component automaton can impose preconditions on each action.

Lemma A.2 *Let $\{A_i\}_{i \in I}$ be a strongly compatible collection of automata, and let $A = \Pi_{i \in I} A_i$. If $\alpha \in execs(A)$ then $\alpha | A_i \in execs(A_i)$ for all $i \in I$. Moreover, the same result holds for fairexecs, scheds, fairscheds, behs and fairbehs in place of execs.*

[7] Note that the second and third components listed are just ordinary Cartesian products, while the first component uses a previous definition.

[8] We use the notation $s[i]$ to denote the i-th component of the state vector s

Certain converses of the preceding lemma are also true. The following lemma says that executions of component automata can be patched together to form an execution of the composition.

Lemma A.3 *Let $\{A_i\}_{i \in I}$ be a strongly compatible collection of automata, and let $A = \Pi_{i \in I} A_i$. For all $i \in I$, let α_i be an execution of A_i. Suppose β is a sequence of actions in $ext(A)$ such that $\beta | A_i = beh(\alpha_i)$ for every i. Then there is an execution α of A such that $\beta = beh(\alpha)$ and $\alpha_i = \alpha | A_i$ for all i. Moreover, if α_i is a fair execution of A_i for all i, then α may be taken to be a fair execution of A.*

Similarly, schedules or behaviors of component automata can be patched together to form schedules or behaviors of the composition.

Lemma A.4 *Let $\{A_i\}_{i \in I}$ be a strongly compatible collection of automata, and let $A = \Pi_{i \in I} A_i$. Let β be a sequence of actions in $acts(A)$. If $\beta | A_i \in scheds(A_i)$ for all $i \in I$, then $\beta \in scheds(A)$. Moreover, the same result holds for fairscheds, behs and fairbehs in place of scheds.*

A.6 Hiding Output Actions

We now define an operator that hides a designated set of output actions in a given automaton to produce a new automaton in which the given actions are internal. Namely, suppose A is an I/O automaton and $\Phi \subseteq out(A)$ is any subset of the output actions of A. Then we define a new automaton, $hide_\Phi(A)$ to be exactly the same as A except for its signature component. For the signature component, we have $in(hide_\Phi(A)) = in(A)$, $out(hide_\Phi(A)) = out(A) \setminus \Phi$, and $int(hide_\Phi(A)) = int(A) \cup \Phi$.

OFFICIAL DISTRIBUTION LIST

DIRECTOR — 2 Copies
Information Processing Techniques Office
Defense Advanced Research Projects Agency
1400 Wilson Boulevard
Arlington, VA 22209

OFFICE OF NAVAL RESEARCH — 2 Copies
800 North Quincy Street
Arlington, VA 22217
Attn: Dr. Gary Koop, Code 433

DIRECTOR, CODE 2627 — 6 Copies
Naval Research Laboratory
Washington, DC 20375

DEFENSE TECHNICAL INFORMATION CENTER — 12 Copies
Cameron Station
Alexandria, VA 22314

NATIONAL SCIENCE FOUNDATION — 2 Copies
Office of Computing Activities
1800 G. Street, N.W.
Washington, DC 20550
Attn: Program Director

HEAD, CODE 38 — 1 Copy
Research Department
Naval Weapons Center
China Lake, CA 93555